Towards Web Security Using PLASMA

Annette Krannig
Fraunhofern-Institute for Computer Graphics IGD

Towards Web Security Using PLASMA

Annette Krannig
Fraunhofer–Institute for Computer Graphics IGD, Germany

Abstract

The World Wide Web is one of the most significant multimedia applications ever developed — and therefore securing the web is one of the most pressing problems. There exist a number of approaches for securing the World Wide Web which, however, usually pursue what one might call a *low level approach* without being able to give adequate consideration to the specific requirements of this multimedia (or hypertext) system.
The subject of this paper is the realization of an adequate security system, which is capable of detecting the different media and structures within hypertext systems and therefore apply different cryptographic mechanisms to them; this resulted in the development of the system PLASMA (**Pla**tform for **S**ecure **M**ultimedia **A**pplications).

PLASMA is a security platform designed within the frame of the Berkom R&D-programme at the Fraunhofer–IGD in Darmstadt whose prototype was developed to provide a means for secure multimedia telecommunications. In order to demonstrate the capabilities of PLASMA, it was integrated into a W3 scenario. The advantages of PLASMA when used in the World Wide Web as well as the architecture created for the integration process are described in the following section.

Keywords

World Wide Web, High level security, Multimedia, Secure communications platform

1 Introduction

Secure telecommunications is a subject which has been addressed extensively in the past; the same is true for the secure World Wide Web. An overview of this topic may be found in the paper by P. Lipp and V. Hassler [10]. Protocol realizations such as SHTTP [14] or SSLeay [13], an implementation of SSL [5], are examples of the prior art in this field.
Yet even in 1996 B. Fernandez noted in [3] that these works concentrated primarily on *low level security*; these approaches do not address the multimedia or structural elements of the application documents. This is an area which has received scant attention in the past and on which this paper tries to shed some light.

From [3]: "Hypertext systems encompass multimedia documents with hyperlinks connecting to other similar documents. Multimedia documents and their hyperlinks may contain a variety of media including audio, video, animation, graphics and text. Extensive research is going on about hypertext/multimedia systems as they are the upcoming platform for worldwide transfer of information. With this rapid growth in hypertext documents and WWW sites, their security is a top priority issue. Compared with other topics, this is a relatively neglected aspect. This is strange because the richness and variety of multimedia information provides with many opportunities to access unauthorized information. Most of the work until now has focused on methods to protect the information or authenticate the users using cryptographic measures. While these are certainly useful, they are too low level to control access based on logical properties of the document information."

To overcome these deficiencies a security platform PLASMA for multimedia telecommunications and applications has been designed and implemented for differentiating between media and structural components and using different cryptographic algorithms accordingly. For the purpose of demonstrating the functionality of PLASMA, the World Wide Web was chosen. The reasons for this were that on one hand more and more transactions occur via the web, making the WWW one of the principal applications in electronic communications overall; on the other hand the World Wide Web constitutes a multimedia system as well as a hypertext system with components which can be distinguished by the structure of the application documents.

PLASMA was a project within the R&D–programme of Deutsche Telekom Berkom GmbH, which is a subsidiary of Deutsche Telekom AG. Deutsche Telekom has developed a product version of PLASMA which is based on Deutsche Telekom's security technology as well as the results of the PLASMA research project. PLASMA is currently available for several operating systems. In this contribution all mentioning of PLASMA refer to the PLASMA research project and the PLASMA prototype realized in this project.

2 New requirements for a secure WWW

The World Wide Web is a multimedia hypertext system; it consists of text documents including images as well as a host of other documents of varying structure and differing media types.

For example, single words or simple buttons in a hypertext document may refer to different documents, images or Java applications which are activated by a mouse click. If the so–called *hyperlink* is activated, the corresponding web server is contacted and the selected *document* is requested by the client. This *document* may be another text document, an image, a Java applet simulating a video or a so–called *form*. *Forms* are sent to the client by the server, filled out by the client and then returned to the server.

In the World Wide Web, text, images, Java applets, forms etc. are transmitted, suggesting the use of different cryptographic mechanisms for these different media types and structural elements. Image and video data may be protected differently from textual data. For example, it is often sufficient to simply reduce the image quality of the image data or to encrypt merely a segment of the image (e.g. encrypting a face). Furthermore, image and video data are of considerably larger volume than textual data which may make the use of faster cryptographic algorithms (which will then be usually less secure) necessary. The divergent media and structures of an application document or the various document types should also be treated differently by a security platform.

Another layer of security is the protection of the entire document during transmission. A document of a multimedia application is usually a composite of different media types; yet it must be considered as a composite whole.

It is necessary to specifically associate media and cryptographic protocols which determine the appropriate algorithm for each media type. However, once the appropriate cryptographic mechanisms for a media type within a document have been determined, they cannot be changed within that document context.

The selection of these security services should be performed by the user since only he can decide on the basis of a specific document whether it should be signed or merely be encrypted in transit — this makes user interactions necessary.

Both are problems which are hardly if at all considered in the current telecommunications systems yet are realized in PLASMA. As a demonstration of these features, PLASMA was integrated into the World Wide Web. The thus developed solution will now be described. The architecture to be presented itself does not offer new insights into the subject of web security but does demonstrate a meaningful application for the security platform PLASMA in the World Wide Web and the new concepts realized within the platform, namely the idea of *high level security*.

Furthermore, this implementation may be used in a real–world scenario for securing communications in the World Wide Web since

- PLASMA allows securing several simultaneous communication sessions; this allows a web server to serve several clients simultaneously using secure connections; similarly a client is capable of starting several simultaneous requests to different servers.

- It is possible to create the presented scenarios (and the architecture presented therein) even when firewalls (of the packet filtering/stateful inspection variety) are used to secure a corporate intranet. PLASMA uses the standard HTTP protocol for tunneling its secure requests and replies thus facilitating real–world secured web communications[4].

- A basic functionality of any kind of cryptographic systems is a correct key management, i.e. the proper administration and storage of the cryptographic keys for each user. The current realization of the key management subsystem in PLASMA conforms to the X.509 authentication framework [15], thus making for a clean implementation of this vital part of a cryptographic system.

3 What is PLASMA?

3.1 Media and structure specific cryptographic operations on application documents

PLASMA is a security platform which is capable of differentiating cryptographically between the various media — the underlying engine also allows different structural components of a document or different document types as they may occur in hypertext documents in the World Wide Web to be operated on using different cryptographic mechanisms.

What are the requirements for this – The *Filter* module The paramount requirement for the above mentioned procedure is that the application (i.e. the multimedia or hypertext system) is capable of handing down information on media structure, structural elements or document types to the security platform. An application document therefore must be split into its multimedia or structural components which in turn must be passed on in PLASMA–specific formats to the security platform. This task is performed by the *Filter*.

The *Filter* is a module that is required by the security platform yet logically resides within the application since only the application has the knowledge required on format structures which vary from application to application and may even diverge strongly from prior versions of the same application. Therefore each application must provide its own PLASMA *Filter*. The data structures within PLASMA for multimedia documents are constant and can be addressed by each application in a similar fashion; it is the application formats which cannot be addressed adequately in a general form by PLASMA.

A *Filter* dissects a multimedia document of the application into its media specific components by noting the locations of subcomponents (e.g. the locations of textual and image parts) into a *Script*; the media specific parts of an application document are thus transformed into their corresponding media specific data objects within PLASMA[1].

If after reverting the cryptographic operations these media specific data objects of PLASMA are to be transformed back into the plaintext document, the *Filter* requires this *Script*. The *Script* provides the required information for the *Filter* to assemble the plaintext document into its original form; the PLASMA specific text objects are written into positions originally containing text portions; similar procedures are followed for image data and other media specific objects within PLASMA. Figure 1 illustrates the concept of a *Filter*.

PLASMA thus obtains media specific data objects from the *Filter* which are then to be processed using various cryptographic mechanisms.

What further modules are required? PLASMA provides three security services for data transmissions in its current implementation; they are *non-repudiation*, *confidentiality* and *integrity*. The object–oriented design of the security platform allows the easy integration of further security services such as the *protection from replay attacks* or the *digital signature creation*[2].

For PLASMA to be able to perform media specific operations on the application document data it is necessary that the security platform can call upon different and media specific protocols for the actual operation; this is realized by the concept of *generic security services*. A generic service *non-repudiation* not only provides one fixed protocol for realizing a non-repudiable component but rather has access to a number of different protocols which then perform the actual operation[3].

The security platform must be capable of activating the particular security services for each media type or structural element and their corresponding cryptographic protocol — these can and should be media specific protocols. The simplest and most obvious approach for this is to implement the corresponding parameters as self–sufficient modules of the platform. This results in the existence of the modules *medium*, *generic security services* and *cryptographic protocols*.

This architecture necessitates another module which creates the relations between these parameters; this relation must express which cryptographic protocol is to be used for a specific medium, structural element or document type with regard to an activated security service. This relation is designated within PLASMA as the *security policy* since this relation actually describes the policy or mode of operation how different media are to be protected cryptographically during transmission — it must in some form be capable of influencing the relations between the other three modules *medium*, *generic security services* and *cryptographic protocols*.

[1]The *Script* is stored together with the other media specific data objects into a *Container* by PLASMA — with the *Container* being a collection object maintaining its subobjects as a list.

[2]The services *non-repudiation* and *digital signature creation* do not differ from a purely technical viewpoint; however, in the latter case the user must be given an opportunity to actively confirm that he wants to sign the given document.

[3]A cryptographic algorithm is referred to herein as a protocol since for example in the case of the DES algorithm the protocol for reverting the encryption on behalf of the recipient is well defined.

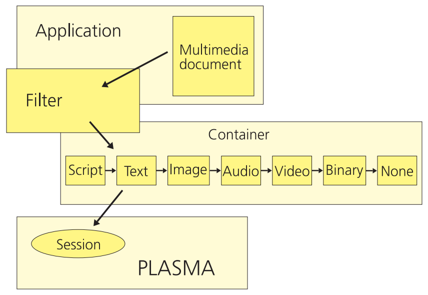


Figure 1: The functionality of a *Filter*

The security policy in PLASMA is stored for each user as an ASCII file in the home directory of each user (*FSP: formal security policy*), thus allowing each user to configure this policy to meet his needs prior to the secured communication itself. The actual use of these security policies in an ongoing communications session, their enforcement and application will be explained later.

The rough outline of the security platform is therefore predetermined to appear similar to Figure 2 at the left side. It is obvious that further modules are required for the implementation of "secure telecommunications" when considering the entirety of a security platform; such objects as the *Session* object depicted in Figure 1 which implements a secure connection between two communications partners within PLASMA — however, a detailed description of the object oriented design of PLASMA was already given in [7] and shall not be repeated.

How are the modules activated? The *generic security services* module could be activated by user activity or alternatively, they might get activated sequentially. The module *cryptographic protocols* must interact with the underlying security technology which provides the actual implementation of the cryptographic algorithms. The *medium* module obtains its data from the *Filter* and differentiate the data stream into several classes. The *security policy* can be set directly by the user or by a security administrator.

3.2 Secure telecommunications using PLASMA

What are the basic requirements for secure telecommunications which are, of course, also realized in PLASMA? First of all each user trying to establish a secure connection to one or multiple communications partners requires access to his personal security–relevant data. These data are stored in a specially secured area, access to which is possible only using a valid PIN.

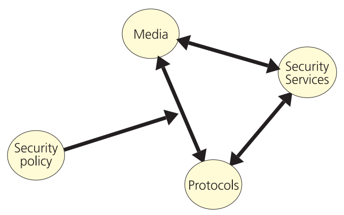
PLASMA requests this PIN upon establishment of a secure connection from the user. If the PIN entered is correct, the storage area (which can best be thought of as a SmartCard) is opened and the user is then enabled to create a secure communications session; otherwise the establishment process is stopped at this point.

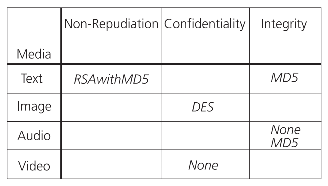
After this access control for the personal security relevant data of the user which in particular contain cryptographic keys it is also necessary for the user to gain certainty about the identity of the communications partner; this requires a mutual authentication of the participating parties in the best case, for which several protocols exist. In PLASMA, this is implemented by means of the X.509 authentication protocol [15].

After a successful X.509 three-way-authentication all participants are well informed about the identity of their counterparts: The web server knows which client wants to access his data and conversely the client knows he has contacted the correct web server — such information is vital, particularly in the case of online transactions and electronic commerce.

During the authentication the security policies, the public asymmetric keys including certificates as well as the session keys which are later used for transferring confidential data are exchanged[4]. The concept of security policies, the exchange thereof in the authentication phase and the subsequent reconciliation of the policies of both communications partners are significant elements of PLASMA: an application document is processed by PLASMA according to the rules extracted from a *Coordinated Security Policy CSP*. The *CSP* is calculated from the security policies (*FSPs*) of the communications partners which are exchanged during the authentication phase.

[4]For the necessary background of cryptographic material refer for example to [15].





Media	Non-Repudiation	Confidentiality	Integrity
Text	*RSAwithMD5*		*MD5*
Image		*DES*	
Audio			*None* *MD5*
Video		*None*	

Figure 2: Relevant modules and a possible security policy

A security policy which yields a relation between the modules medium/structure, cryptographic protocols and security services, thus assigning a cryptographic protocol to a combination of a medium and an activated security service is partitioned into two segments: the *profile* and the *rule* part.

The profile part lists the cryptographic protocols available for each security service; this is useful since it may be the case that one of the communicating partners is willing only to communicate if SmartCards are used for storing the cryptographic keys or even that one side wants to exclude a simplistic cryptographic algorithm from the negotiation since it is deemed too insecure by that party.

The rule part lists the specific cryptographic protocols which are to be used when encountering a given media or structural type of an application document in order to provide a requested security service[5] (cf. Figure 2, right side).

These security policies are created locally at each user's site and must be reconciled prior to the communication proper and immediately after the mutual authentication with communications partner; result is the *CSP*, mentioned before. To achieve this the set intersection of the profile parts of both policies is generated; in the case of the rule parts the set union is created. This way only such protocols are allowed for the communication which are available on both sides of the communications channel and only those cryptographic protocols are selected which do not contradict any rule from either party.

After a successful authentication of the communications partners the secure communication itself between the participating parties may commence. To achieve secure communications, the security services confidentiality, non-repudiation and integrity are made available by PLASMA[6].

Using PLASMA it is now possible to send messages with guaranteed integrity, i.e. the message will arrive provably at the recipient as it has been sent by the sender; non-repudiable message passing means that the sender of the message is uniquely identifiable by the recipient – beyond that the non-repudiation automatically includes the integrity of the transmitted document; lastly, PLASMA is capable of generating confidential messages, i.e. messages which only authorized parties – the sender and the recipient – are able to decrypt; all other parties can only access the message in encrypted form.

The concept of security polices and, correspondingly, the media–specific operations which also include the structural properties of an application document are now brought to the fore in the following: the generic security services are activated by default sequentially by PLASMA or after explicit user interaction — only the user may decide at runtime whether a document to be transmitted shall be signed or protected against a loss of confidentiality.

Consequently, a media specific cryptographic protocol matching the activated security service and current media type must be selected which will then be used to meet the security policies' goal. To achieve this, the security policy *CSP* is queried; this object is structured as a table assigning each media type and generic security service a specific cryptograpic protocol (cf. Figure 2 at the right side).

The result of this query is a cryptographic protocol which is subsequently activated by PLASMA. The data are transmitted to this protocol which will then perform the cryptographic operations specific to itself on these data.

Lastly, an integral part of each secure communications session is the ability to dismantle such a connection properly in such a way as to disallow a possible intruder the disruption of such a connection. Therefore this functionality has been integrated into PLASMA.

The possibility of user interaction is, similar to the media specific operation on an application document, only possible if the security platform is situated logically close to the application and is capable of communicating with the

[5]The "protocol" *None* is for document parts which should not be treated cryptographically.

[6]Through the object oriented design of the security platform PLASMA, as described in [7], it is easily possible to integrate further generic security services into the platform.

application. This goal has been reached within PLASMA by taking the approach of locating the platform close to the application based on the concept of the GSS-API [9], [17].

An essential feature of any kind of security (in the cryptographical sense) is the need for key management, i.e. the proper administration and storage of the cryptographic keys for each user. PLASMA is based on a security technology which implements the cryptograpic algorithms (cf. [7]). The implementation used as a security technology in PLASMA is SecuDe (cf. [16]); it is also possible to implement it using a different security technology, for example utilizing the widely spread security package PGP [18].

Therefore the key management of PLASMA is fundamentally dependent on the functionality of the underlying security technology. If these security technologies offer key management themselves, PLASMA can merely integrate it into itself; otherwise it needs to be modeled within the platform itself; the security technologies PGP and SecuDe for example offer proprietary utilities for certification of their keys and also define the makeup of certified keys and certificates[7].

4 The integration of PLASMA into the World Wide Web

There are two well–distinguished layers in this W3 scenario. The first layer is that of the World Wide Web and is situated between the browser, the proxy and the CGI programs. In this layer, the client poses requests to the server by activating a hyperlink; this hyperlink contains a call to a CGI program which creates the server's response: a HTML page which in turn contains a hyperlink to the next CGI program in the sequence; this is the layer of the HTTP protocol.

The second layer is located between the PLASMA applications on the client and server sides. Here, data are passed through to the security platform so that PLASMA may perform cryptographic operations on them. The protocol to be used between the PLASMA applications is predetermined by the application independent API of PLASMA and may be considered separable into three phases: The connection establishment and authentication phase, the secure document transmission phase, and the connection teardown phase.

When integrating PLASMA into the WWW the principal problem is to pass the data on both sides from the WWW layer onto the PLASMA layer. On the serverside this is possible using CGI programs (Common Gateway Interface [12]). When using a Mosaic browser, a CCI implementation (Common Client Interface [11]) on the client side and CGI programming on the server side would be possible; but here it is not possible to encrypt a client's request[8]: The client formulates his request by activating a hyperlink to a server side page. This causes the service request to be forwareded immediately to the server. In *addition* to that, the request can also be transmitted to the CCI interface "in parallel" (and therefore to PLASMA) – but by then it is already too late to secure the request. A *PlugIn* is another possibility for a browser to pass data to another program and to read back the response from this program. The PlugIn API is a proprietary extension of the Netscape Navigator API and is not supported by other browsers.

The PLASMA relevant data are embedded into the HTTP protocol; to fulfill the requirements for browser independent communications there is to find a way to pass the information onto PLASMA on the way from client to server and conversely, i.e. to filter the data for the security platform. Therefore it is not possible to use a special propose filter module because HTTP is not aware of the filter.

The proxy, however, is well known of the HTTP protocol. Proxies in W3 buffering the informations from the server on clientside and handing over these data to the browser if they are completely arrived. So the proxy is the component where the filter functionality can be integrated. Therefore PLASMA has been integrated into the World Wide Web using a proxy on the client side and using CGI programs on the server side. This allows a secure usage of the World Wide Web by all clients, no matter what the browser of their choice might be (Netscape, Mosaic,. . .).

The data to be passed on to PLASMA can only be transferred on WWW via the HTTP protocol. Informations intended for PLASMA are so-called PLASMA tokens. There exists three different kinds of PLASMA token; the PLASMA application has to discern whether the token is an authentication token (type `X509`), a document or *Container* token (type `Cont`), or a *Final* token (type `Finl`). It must be possible in any phase of the protocol between the PLASMA applications to pass data from the WWW layer onto PLASMA:

[7]The certificate structures used in SecuDe comply with the X.509 authentication framework [15] which requires the existence of certification authorities for the certification of the asymmetric public keys.

[8]The Mosaic browser family offers a feature to access programs such as security platforms directly from the web client. To achieve this goal, the CCI was defined for the client side and on the server side the CGI specification was established.

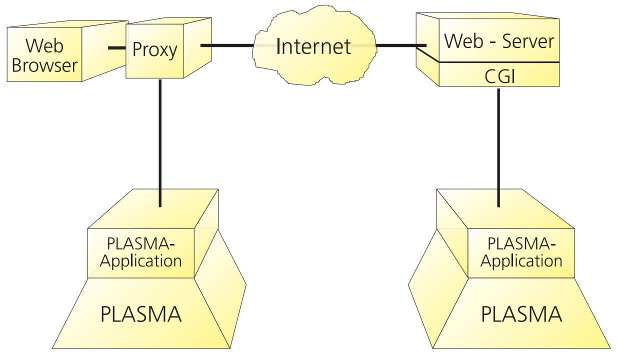


Figure 3: Overview of the implemented proxy architecture

- In the authentication phase PLASMA on client side must be able to retrieve the requests for embedding the relevant PLASMA token into the PLASMA packet.
- In the document transmission phase cryptographic operations must be performed on the informations regarding which document the client or user requested; therefore all hyperlinks of the previously sent pages must be embedded in PLASMA packets.
- On the server side the CGI programs detect in the requests the embedded PLASMA tokens so they may be passed on to the PLASMA application.
- Subsequently they retrieve a HTML page including a hyperlink to the next CGI program in sequence as well as the corresponding PLASMA token; this token must then be passed to the PLASMA application on the client side.

Because these PLASMA tokens can only transferred on the WWW, they are transferred connected to elements of the HTTP protocol, i.e. the replies of the server and the requests of the clients. These HTTP elements include comments field not considered by the HTTP protocol which can be used to fill in the identification tag for the filtering process. These packages will be called in the following as PLASMA packets. PLASMA packets contain consequently an identification tag which the CGI programs and the proxy may detect.
The communication between web client and web server is not diverging from the normal HTTP protocol, therefore the architecture presented here can also be performed with a firewall. It is, however, important to note that the proxy used as a filter should not be the same proxy used for protecting a domain by a firewall. For a secure web scenario every client user has to start the proxy on this machine, where he started his own PLASMA system; every user should start in the best case his own proxy.

4.1 The components of the proxy architecture

The following section deals with the components of the secure World Wide Web architecture using PLASMA and a proxy and describes their responsibilities. The separate components are PLASMA itself as a server process on the client and server side, an application of PLASMA on both sides which is capable of calling the application independent interface functions, the proxy on the client side and the CGI programs on the server side which transfer the data from the web server in their defined states to PLASMA. An overview of the components used in the proxy architecture is given in Figure 3.

PLASMA as a server process The "philosophy" of the World Wide Web can be summed up as follows: a client establishes a channel to a WWW server using an URL request. The server responds to this request by sending the requested information back to the client (if possible) and closing the communications channel immediately thereafter. If the client wants to direct another request to the server, a new communications channel must therefore be established.

World Wide Web servers can call external programs via the CGI, for example to perform database accesses or to activate security software. The CGI specification uses a simple scheme for such calls; the CGI program must terminate once it has processed the data and returned it to the web server. The server in turn passes the data generated by the CGI program on to the client and "forgets" everything about the transaction. Under normal circumstances this should not be a problem – unless a CGI program needs to maintain state between two calls.

If one wants to use PLASMA to secure web communications on the server side, this means that, for example, the server sends the request for the first authentication token to PLASMA via a CGI call. After serving this request, the CGI program terminates, therefore PLASMA terminates as well. All data that would have to be stored to maintain a secure link between client and server are then lost (this includes public keys and certificates as well as session keys etc.).

Therefore PLASMA must run permanently on the server side and it must not terminate once the CGI application terminates when returning data to the web server. The PLASMA system is therefore turned into a daemon process; this allows it to maintain state information for several communications links across several CGI calls within PLASMA[9]. Similar considerations apply to the client side; since the proxy also maintains only transient processes for each request, the state information for a secure communications link must be maintained in a PLASMA application as well.

The CGI programs As has been mentioned before, World Wide Web servers are able to adress external programs via the CGI. The call of a CGI program is usually initiated by activating a hyperlink that was found in a HTML document previously transferred to the client side. Since the activities of the CGI programs are well defined within the web scenario for each state of the protocol between sender and receiver, separate CGI programs were defined for each state.

Both CGI calls during the authentication phase are performing identical tasks with regard to PLASMA, they do, however, send different web pages as a response to the client.

The CGI program called during document transfer gets the requested document and decides whether or not the client has access to the requested document by means of the embedded client *DName*[10]. Lastly, there is a separate CGI program which gets called if a request for terminating the *session* with the client is detected.

All CGI programs are searching for the PLASMA identification string and passing as the proxy on client side these data onto PLASMA.

The proxy From the security perspective the proxy is merely a filter deciding whether or not data must be passed through PLASMA in this architecture; it searches for a PLASMA identification tag in each data packet it receives. Data containing the PLASMA identification tag are passed on to PLASMA, otherwise the proxy passes the data on to the browser or the server, respectively, without interacting with PLASMA.

In order to maintain the full functionality of a "normal" proxy while embedding the necessary functionality, a CERN HTTPD 3.0 was used as a base for embedding the PLASMA modifications. Porting these modifications to other proxy architectures should be fairly easy since these are well embedded. The drawback inherent in this approach is the necessity to follow standard operating procedures, i.e. each browser request must be followed by a response from the server.

Since the browser knows nothing of the security enhancement of the communications channel, all transactions must be performed between the proxy and the server. Each request for PLASMA services must contain such a string which allows the proxy to intercept it and send it to the PLASMA application; the same goes for incoming data from the server; therefore the identification string is inserted by the CGI program.

The PLASMA applications on client and server side The PLASMA applications are the agents tasked with accepting transferred data from the proxy on the client side and from CGI programs on the server side. It passes these data on to PLASMA and returns the output to the proxy or to the CGI programs for secure transfer across the public network. The PLASMA application gets passed only such data packets which actually have to be modified cryptographically by PLASMA in either the "to" or "from" direction.

It is the responsibility of the PLASMA application to analyze received data packets and to pass these PLASMA tokens on to the security platform using the correct interface functions of the application independent interface [8].

Since PLASMA is required to operate permanently as a background process, socket connections between the proxy and the PLASMA application on the client side as well as between the CGI programs and the PLASMA server application are necessary.

4.2 The dynamic model

The following section details the interactions between the separate components required for secure web communications. The dynamic model is subdivided into three phases: the authentication phase, the secure transmission phase and the connection teardown phase.

[9] PLASMA on both sides is capable of securing several simultaneous communications, so a web server is able to serve several clients simultaneously and similarly a client is capable of starting several simultaneous requests to different servers using secure connections.

[10] *DName*s are unique identifiers of the participating parties which are defined in the X.500 standards suite (cf. [15]).

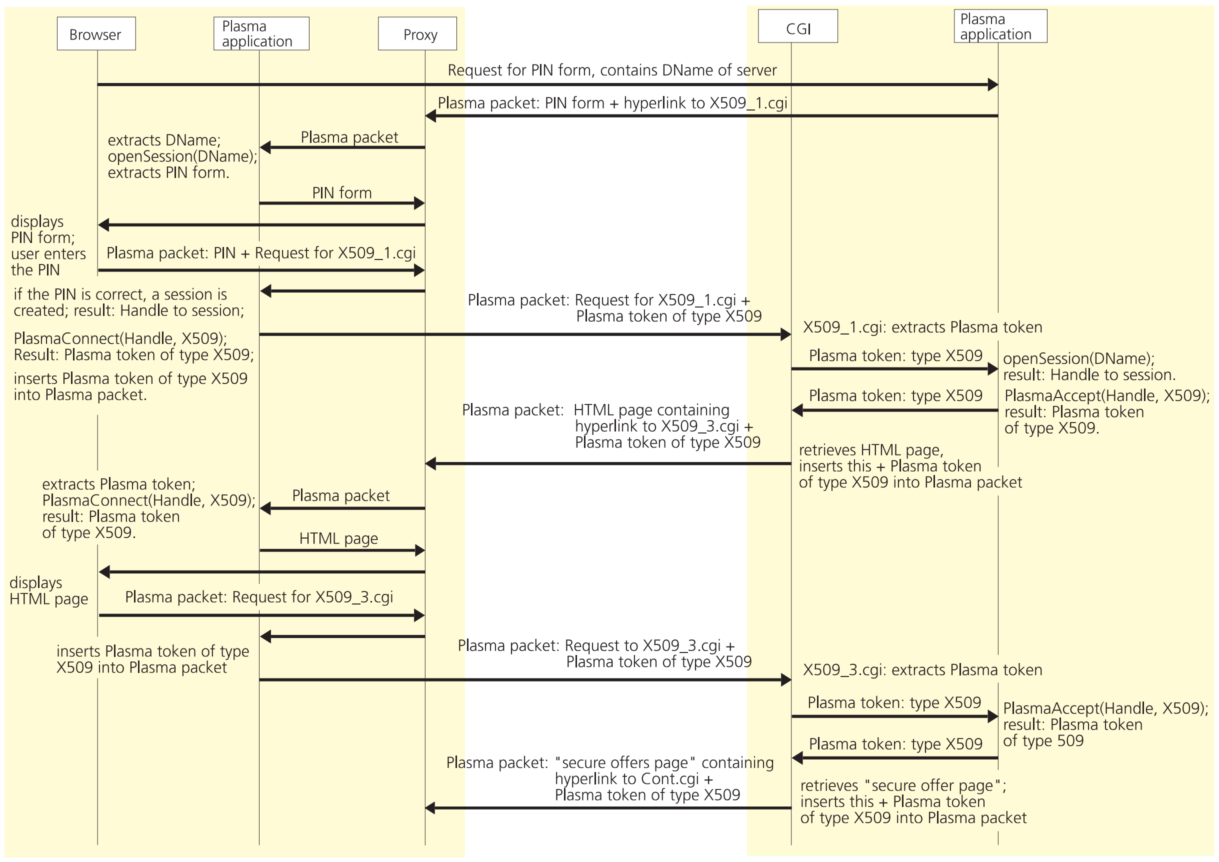

Browser
Plasma application
Proxy
CGI
Plasma application
Request for PIN form, contains DName of server
Plasma packet: PIN form + hyperlink to X509_1.cgi
Plasma packet
extracts DName; openSession(DName); extracts PIN form.
PIN form
displays PIN form; user enters the PIN
Plasma packet: PIN + Request for X509_1.cgi
if the PIN is correct, a session is created; result: Handle to session;
Plasma packet: Request for X509_1.cgi + Plasma token of type X509
PlasmaConnect(Handle, X509); Result: Plasma token of type X509;
inserts Plasma token of type X509 into Plasma packet.
X509_1.cgi: extracts Plasma token
Plasma token: type X509
openSession(DName); result: Handle to session.
Plasma token: type X509
PlasmaAccept(Handle, X509); result: Plasma token of type X509.
Plasma packet: HTML page containing hyperlink to X509_3.cgi + Plasma token of type X509
retrieves HTML page, inserts this + Plasma token of type X509 into Plasma packet
extracts Plasma token; PlasmaConnect(Handle, X509); result: Plasma token of type X509.
Plasma packet
HTML page
displays HTML page
Plasma packet: Request for X509_3.cgi
inserts Plasma token of type X509 into Plasma packet
Plasma packet: Request to X509_3.cgi + Plasma token of type X509
X509_3.cgi: extracts Plasma token
Plasma token: type X509
PlasmaAccept(Handle, X509); result: Plasma token of type 509
Plasma token: type X509
Plasma packet: "secure offers page" containing hyperlink to Cont.cgi + Plasma token of type X509
retrieves "secure offer page"; inserts this + Plasma token of type X509 into Plasma packet

Figure 4: The interactions in the authentication phase

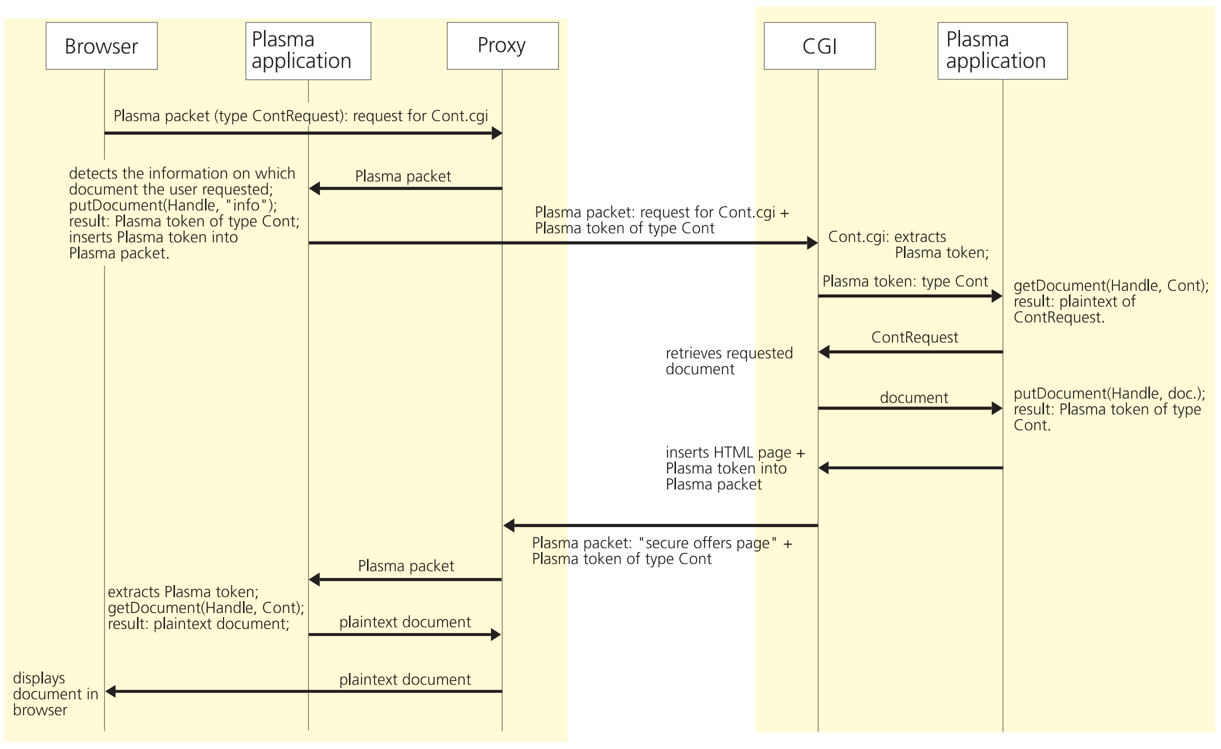
The authentication phase The authentication phase commences by the client establishing a secured connection with the server within PLASMA; to achieve this the application on client side calls the PLASMA C–API function `openSession()`. For this the *DName* of the server is required which is inquired in the first request. The server's response contains the *DName* as well as a PIN form into which the user at the client side will enter his PIN.

Once the `Session` has been opened successfully, the client calls the authentication function for the sender side for the first time (`PlasmaConnect()`). The result is a PLASMA token of the type `X509` which is then sent to the server.

Upon receiving this PLASMA token (`X509`) at the server side, a *session* is created within PLASMA and the authentication towards the client is continued (first call of the authentication for the receiver side (`PlasmaAccept()`).

The exact flow of the protocol for authentication may be gleaned from Figure 4. In this phase first the PIN form including the server *DName* will be transmitted to the client. Next the CGI programs `X509_1.cgi` and `X509_3.cgi` are called on the server side, which are sending different HTML pages to the client side; the first HTML page is only used for sending the hyperlink to the next CGI program (`X509_3.cgi`), the next page already present the offers of the server.

In this state only PLASMA token of type `X509` are transmitted – both applications have appropriate knowledge as to which API calls need to be performed when receiving PLASMA tokens of this type.


Browser
Plasma application
Proxy
CGI
Plasma application
Plasma packet (type ContRequest): request for Cont.cgi
Plasma packet
detects the information on which document the user requested; putDocument(Handle, "info"); result: Plasma token of type Cont; inserts Plasma token into Plasma packet.
Plasma packet: request for Cont.cgi + Plasma token of type Cont
Cont.cgi: extracts Plasma token;
Plasma token: type Cont
getDocument(Handle, Cont); result: plaintext of ContRequest.
ContRequest
retrieves requested document
document
putDocument(Handle, doc.); result: Plasma token of type Cont.
inserts HTML page + Plasma token into Plasma packet
Plasma packet: "secure offers page" + Plasma token of type Cont
Plasma packet
extracts Plasma token; getDocument(Handle, Cont); result: plaintext document;
plaintext document
plaintext document
displays document in browser

putDocument(Handle, Data,...) := plaintext data are passed on to Plasma for cryptographic operations,

getDocument(Handle, Data,...) := encrypted data are passed on to Plasma for reconstruction;

ContRequest: this Plasma packet contains the call of the CGI program Cont.cgi as well as information on which document the client requested.

Figure 5: Interactions in the data transmission phase

The document transmission phase After a successful three-way-authentication requests and replies between the web client and server may be transmitted securely. The server offers its documents via the

secure offers page (cf. Figure 5); on the client side the user selects a document and activates the corresponding CGI program upon confirmation. The call of the CGI program is not encrypted; however, the information regarding which document has been requested by the user must be encrypted at the client side[11].

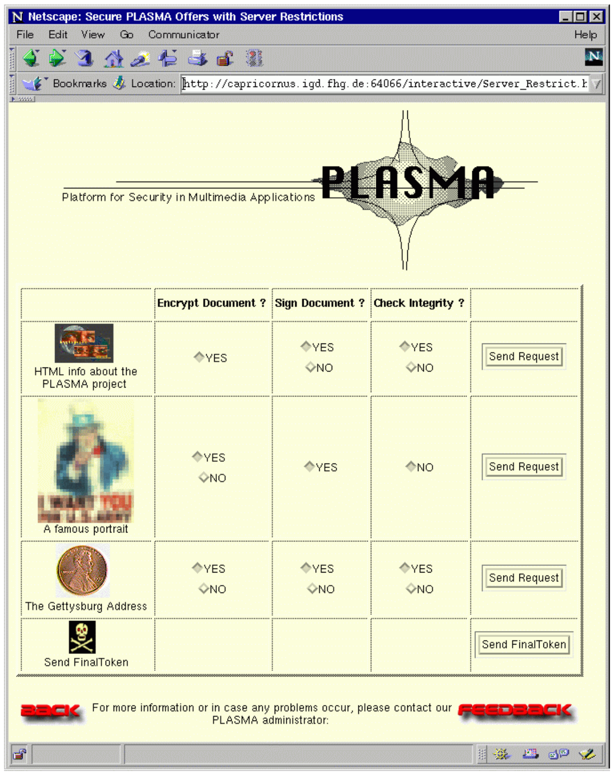
In this phase only tokens of the type `Cont` are transmitted; the PLASMA application is wired to call the filter function `getDocument()` for reverting the cryptographic operations on this token type. In this phase plaintext data are passed onto PLASMA for cryptographic operations using the `putDocument()` API function.

The CGI program `Cont.cgi` gets called on the server side. It decrypts the information on the requested document and encrypts the document itself prior to transmission. The HTML pages sent in this phase represent the documents or offers of the server; the exact protocol is shown in Figure 5.

Connection teardown All HTML pages of a server allowing secured communcations using PLASMA which offer documents contain a hyperlink to the CGI program `Finl.cgi` – refer to Figure 6 at the bottom. Upon activation of this hyperlink the *session* at the client side is torn down by calling the API function `closeSession()` of PLASMA[12].

[11]This requires the application on the client side to detect the condition that the API function `putDocument()` for cryptographic operations in “to” direction must be called, therefore the request type *ContRequest*, which is also a PLASMA packet, was introduced.

[12]This requires the application on the client side to detect the condition that the API function for conncection shutdown must be called, therefore the request type *FinlRequest* was introduced, also a PLASMA packet.


Netscape: Secure PLASMA Offers with Server Restrictions
File Edit View Go Communicator Help
Bookmarks Location: http://capricornus.igd.fhg.de:64066/interactive/Server_Restrict.h
PLASMA
Platform for Security in Multimedia Applications

	Encrypt Document ?	Sign Document ?	Check Integrity ?	
HTML info about the PLASMA project	◇YES	◇YES ◇NO	◇YES ◇NO	Send Request
A famous portrait	◇YES ◇NO	◇YES	◇NO	Send Request
The Gettysburg Address	◇YES ◇NO	◇YES ◇NO	◇YES ◇NO	Send Request
Send FinalToken				Send FinalToken

BACK For more information or in case any problems occur, please contact our PLASMA administrator: FEEDBACK

Figure 6: Interaction opportunities for the user in the web demo

The result of this connection teardown is a PLASMA token of the type `Finl` which now must be sent to the server. The application on the server side also recognizes that the *session* is to be disconnected upon receiving this token of type `Finl` and thus also calls `closeSession()`.

5 Summary

In conclusion the results of the integration of the PLASMA security platform into the World Wide Web are summed up — these results are made possible by the implementation of the idea of *high level security* within PLASMA.
The just presented architecture allows a mutual authentication of web clients and servers which may also be achieved by means of other security platforms; after a successful authentication the server knows which client user wants to obtain data from him and may use this identification by means of appropriate CGI programs to grant or deny access (access control to the requested documents).
Requests and responses may now be secured — by locating the security platform close to the application the interactions of the user may be considered, i.e. the server can offer its documents in such a way as to allow the client to determine whether the document is to be sent confidentially, non-reputiably or with integrity checks. Furthermore the server may predetermine which documents need to be protected using specific security services; the web page in Figure 6 exemplifies the opportunities for interaction with the user; the second image from the top may be treated confidentially and be transmitted non-reputiably; interaction with the user is possible since the generic security services within the platform have been realized as independent modules – these merely have to be activated by a mouse click via the interface of PLASMA.

Finally, and this is the most significant feature facilitated by using PLASMA, the different media and different structures of the HTML documents may be differentiated cryptographically. Using PLASMA it is possible to ensure that forms, which may be containing a contract, will always be signed – both by the server to make sure that the client knows he is transferring his credit card number to an authorized server as well as by the client side to make sure that the client user has actually signed the “contract” in that particular form; using PLASMA it is possible that for

example textual data is always integrity protected and images are protected against loss of confidentiality during transmission.
Finally it ensures that an unauthorized disruption of a session by a client or server can be detected.

6 Acknowledgements

I would like to thank everyone who has contributed to the design of the just presented architecture for their insights and useful discussions; in particular I would like to thank Xiaodong Liu for the invaluable support in evaluating peculiar problems of the WWW and for aid in the specification of the secure WWW architecture. I would like to thank Dr. Eckhard Koch for the opportunity to realize this project. Furthermore I would like to extend particular gratitude to Xiaodong Liu again, Henning Daum, Stephen Wolthusen and Ingo Jankowski for extremely productive assistance in implementing these system, as well as Dr. Christoph Busch for proofreading.
A different approach for implementing the web security scenario using the CCI system with a Mosaic browser and a different underlying security technology platform was developed by Cheng Dong, Mehrdad Jalali and Jian Zhao, also at the Fraunhofer-Institute; this system has provided several crucial insights into the particular problems one encounters when securing the web.

References

[1] T. Berners-Lee, A. Luotonen, H. F. Nielsen, A. Secret (1994) The World–Wide–Web. *Communications of the ACM, Vol.37 No. 8*.

[2] H. Cheng, X. Li (1996) On the application of image decomposition to image compression and encryption. *Chapman & Hall, Communications and Multimedia Security II, ed. P. Horster, 116-127*.

[3] B. Fernandez, R. Nair, M. Larrondo-Petrie, Y. Xu (1996) High–Level Security Issuses in Multimedia/ Hypertext Systems. *Chapman & Hall, Communications and Multimedia Security II, ed. P. Horster, 13-24*.

[4] R. Fielding, J. Gettys, J. Mogul (1996) Hypertext Transfer Protocol – HTTP/1.1. *IETF draft*.

[5] A. O. Freier, P. Karlton, P. C. Kocher (1996) The SSL Protocol Version 3.0. *Netscape Communications Corporation*.

[6] M. Gehrke, E. Koch (1992) A Security Platform for Future Telecommunication Applications and Services. *Proc. of the 6th Joint European Networking Conference*.

[7] A. Krannig (1996) PLASMA - Platform for Secure Multimedia Applications. *Chapman & Hall, Communications and Multimedia Security II, ed. P. Horster*.

[8] A. Krannig, H. Daum (1996) PLASMA — The Application Independent API. *Fraunhofer–IGD Darmstadt, Technical Report*.

[9] Linn, J. (1993) *RFC 1508 – Generic Security Service Application Programming Interface*

[10] P. Lipp, v. Hassler (1996) Security concepts for WWW. *Chapman & Hall, Communications and Multimedia Security II, ed. P. Horster*.

[11] NCSA (1996) CCI Specification. *http://www.ncsa.uiuc.edu/SDG/Software/ XMosaic/CCI/cci-spec.html*.

[12] NCSA (1996) CGI The Common Gateway Interface. *http://hoohoo.ncsa.uiuc.edu/cgi/*.

[13] H. Reif (Juni 1997) Secure Socket Layer: Chiffrieren und Zertifizieren mit SSLeay. *IX Multiuser Multitasking Magazin*.

[14] E. Rescorla, A. Schiffman (1996) The Secure HyperText Transfer Protocol. *IETF draft*.

[15] B. Schneier (1996) Applied Cryptography, 2nd ed. *Wiley*.

[16] W. Schneider (1993) SecuDe: Overview. *GMD–TKT Darmstadt*.

[17] Wray, J. (1993) *RFC 1509 – Generic Security Service API : C Bindings*

[18] P. Zimmermann et al. (1993) PGP. *Phil Zimmermann*.